

## Density Clustering Algorithm Based on Radius of Data (DCBRD)

<sup>1</sup>A.M. Fahim, <sup>2</sup>A.M. Salem, <sup>3</sup>F.A. Torkey, <sup>4</sup>M.A. Ramadan  
<sup>1</sup>Faculty of Education, Suez Canal University ahmedfahim@yahoo.com  
<sup>2</sup>Faculty of Computers & Information Ain Shams University  
<sup>3</sup>Faculty of Computers & Information Minufiya University  
<sup>4</sup>Faculty of Science Minufiya University

### **Abstract**

*Clustering algorithms are attractive for the task of class identification in spatial databases. However, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to determine the input parameters, discovery of clusters with arbitrary shape and good efficiency on large databases. The well-known clustering algorithms offer no solution to the combination of these requirements. In this paper, a density based clustering algorithm is presented relying on a knowledge acquired from the data which is designed to discover clusters of arbitrary shape. The proposed algorithm requires no input parameter. We performed an experimental evaluation of the efficiency of it using real and synthetic data. The results of our experiments demonstrate that the proposed algorithm is significantly efficient in discovering clusters of arbitrary shape and size.*

### **1. Introduction**

Large datasets have been collected or produced in many application domains, such as bioinformatics, physics, geology, and marketing, and some have reached the level of terabytes. Since the knowledge hidden in this data is usually of great strategic importance[10]. One of the primary data analysis tasks is the cluster analysis, which help user to uncover the knowledge hidden in the collected data. Clustering is one of the most important tasks in data mining and knowledge discovery[5]. The main goal of clustering is to organize a set of objects into groups such that the objects in the same group are similar to each other and different from those in other groups. Clustering groups database data into meaningful subclasses in such a way that minimizes the intra-differences and maximizes the inter-differences of these subclasses[17]. Clustering technique is applied in many areas, such as statistical data analysis, pattern recognition, image processing, and other businesses applications. Up to now, many clustering algorithms have been proposed, each of these algorithms have drawbacks and advantages. A clustering algorithm is considered to be good if it satisfies the following requirements, (1) minimal requirements of domain knowledge to determine the values of its input parameters, which is very important problem especially for large data sets. (2) Discovery of arbitrary shaped clusters. (3) good efficiency on large data sets, data set may contains large number of objects or the object described by large number of features or data is large in both previous dimension. The well-known clustering algorithms offer no solution to the combination of these requirements.

In this paper we propose a clustering algorithm based on knowledge acquired from the data set, and apply the main idea of density based clustering algorithms like DBSCAN. The proposed algorithm will be called density clustering based on radius of data (DCBRD). The DCBRD algorithm requires no input parameters, discovers arbitrary size and shaped clusters, is efficient even for large data sets especially data with large dimension. The paper is organized as follows. In section 2, some previous clustering algorithms are discussed. In section 3, simplified review about DBSCAN algorithm is presented. In section 4, the overlapped circular regions are presented, and we show how these circles are created and how the algorithm use them. Section 5 demonstrates the proposed algorithm. We describe the experimental results in section 6 and conclude with section 7.

## 2. Related work

There are many clustering algorithms proposed, these algorithms may be classified into partitioning, hierarchical, density and grid based methods[8]. The first two types are the most common. *Partitioning algorithms* construct a partition of a database  $D$  of  $n$  objects into a set of  $k$  clusters, where  $k$  is an input parameter for these algorithms, refers to the required number of clusters or the number of clusters contained in the database if the value of  $k$  is true. All partitioning algorithms start with an initial partition of  $D$  and then use an iterative control strategy to optimize an objective function which is typically average mean squared error (MSE). Each cluster is represented by the gravity center of the cluster in k-means algorithm[12] or by the most centrally located object in k-medoid algorithms[11],[13]. Consequently, partitioning algorithms use a two-step procedure. First, determine  $k$  representatives minimizing the objective function (MSE). Second, assign each object to the cluster with the closest representative. Thus, the shape of all clusters found by a partitioning algorithm is convex.

Hierarchical algorithms create a hierarchical decomposition of a database  $D$ . The hierarchical decomposition is represented by a dendrogram (tree) that iteratively splits  $D$  into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of  $D$ . The dendrogram can either be created from the leaves up to the root (agglomerative approach) or from the root down to the leaves (divisive approach) by merging or dividing clusters at each step. Hierarchical algorithms do not need  $k$  as an input. However, a termination condition has to be defined indicating when the merge or division process should be terminated.

The basic hierarchical clustering algorithm works as in[6]. Initially, each object is placed in a unique cluster. For each pair of clusters, the values of dissimilarity or distance are computed. For instance, the distance may be the minimum distance of all pairs of points from the two clusters (single-link method)[14], or the distance may be the maximum distance of all pairs of points from the two clusters (complete-link method) [2], [9], or the distance may be the average distances of all pairs of points from the two clusters (average-link method) [16]. In every step, the clusters with the minimum or maximum or average distance in the current clustering are merged until all points are contained in one cluster or termination condition satisfied.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [18] proposes a special data structure to condense information about subclusters of points. A Clustering Feature (CF) is a triple that contains the number of points, the linear sum and the square sum of all points in the cluster. Clustering features are organized in a height balanced tree. This algorithm may not work well when clusters are not “spherical” because it uses the concept of radius or diameter to control the boundary of a cluster. Another hierarchical clustering is the algorithm CURE that has been proposed in [7]. This algorithm stops the creation of a cluster hierarchy if a level consists of  $k$  clusters where  $k$  is one of several input parameters. It utilizes multiple representative points to evaluate the distance between clusters, thereby adjusting well to arbitrary shaped clusters and avoiding the chain effect but this algorithm may not work well when data object described with large number of attributes because it depends on kd-tree.

Density-Based Clustering algorithms group objects according to specific density objective functions. Density is usually defined as the number of objects in a particular neighborhood of a data objects. In these approaches a given cluster continues growing as long as the number of objects in the neighborhood exceeds some parameter. Clusters can be thought of as regions of high density, separated by regions of no or low density. The most popular one is probably DBSCAN (Density-Based Spatial Clustering of Applications with Noise)[3]. DBSCAN finds arbitrary-shaped clusters, it requires the user to specify the radius of the neighborhood and the minimum number of objects it should have; optimal parameters are difficult to determine. DBSCAN employs a spatial index(R\*-tree) [1] to help finding neighbors of a data point. Thus, the complexity is improved to  $O(n \log n)$  as opposed to  $O(n^2)$  without the index. The performance degrades for high dimensional data since R\*-tree not works well as dimensional increase. In this paper new index structure is proposed. This structure also used to obtain an optimal value for the radius of the

neighborhood. The proposed algorithm is based on this structure and the main idea of DBSCAN algorithm.

### 3. DBSCAN Algorithm

The key idea of density-based clustering is that for each object of a cluster the neighborhood of a given radius ( $Eps$ ) has to contain at least a minimum number of objects ( $MinPts$ ), i.e. the cardinality of the neighborhood has to exceed some threshold. We will first give a short introduction to DBSCAN.

**Definition 1:** (directly density-reachable) An object  $p$  is *directly density-reachable* from an object  $q$  wrt.  $Eps$  and  $MinPts$  in the set of objects  $D$  if :

- 1)  $p \in N_{Eps}(q)$ , ( $N_{Eps}(q)$  is the subset of  $D$  contained in the  $Eps$ -neighborhood of  $q$ ).
- 2)  $Card(N_{Eps}(q)) \geq MinPts$ .

**Definition 2:** (density-reachable) An object  $p$  is *density-reachable* from an object  $q$  wrt.  $Eps$  and  $MinPts$  in the set of objects  $D$ , denoted as  $p > Dq$ , if there is a chain of objects  $p_1, \dots, p_n$   $p_1 = q, p_n = p$  such that  $p_i \in D$  and  $p_{i+1}$  is directly density-reachable from  $p_i$  wrt.  $Eps$  and  $MinPts$ .

**Definition 3:** (density-connected) An object  $p$  is *density-connected* to an object  $q$  wrt.  $Eps$  and  $MinPts$  in the set of objects  $D$  if there is an object  $o \in D$  such that both  $p$  and  $q$  are density-reachable from  $o$  wrt.  $Eps$  and  $MinPts$  in  $D$ .

Figure 1 illustrates the definitions on a sample database of objects from a 2-dimensional vector space.

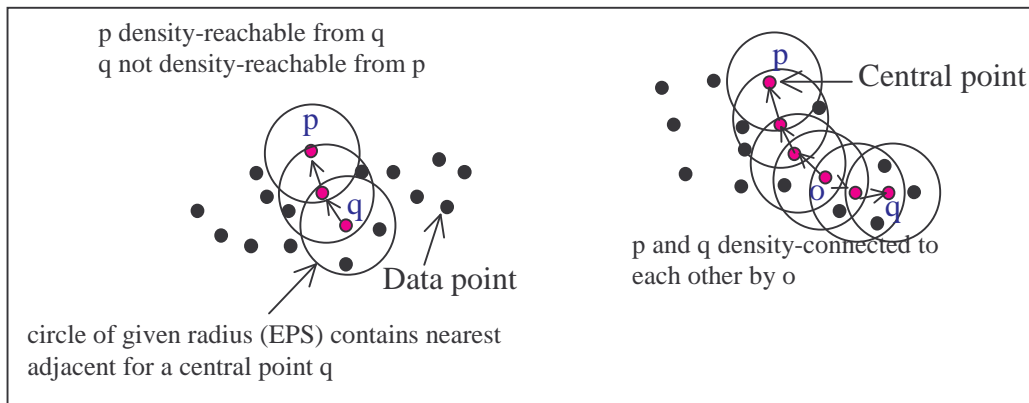


Figure 1: Density-reachability and density-connectivity

**Definition 4:** (cluster) Let  $D$  be a set of objects. A *cluster*  $C$  wrt.  $Eps$  and  $MinPts$  in  $D$  is a non-empty subset of  $D$  satisfying the following conditions:

- 1) Maximality:  $\forall p, q \in D$ : if  $p \in C$  and  $q > D p$  wrt.  $Eps$  and  $MinPts$ , then also  $q \in C$ .
- 2) Connectivity:  $\forall p, q \in C$ :  $p$  is density-connected to  $q$  wrt.  $Eps$  and  $MinPts$  in  $D$ .

**Definition 5:** (noise) Let  $C_1, \dots, C_k$  be the clusters wrt.  $Eps$  and  $MinPts$  in  $D$ . Then, we define the *noise* as the set of objects in the database  $D$  not belonging to any cluster  $C_i$ , i.e.  $noise = \{p \in D \mid \forall i: p \notin C_i\}$ .

```

Algorithm DBSCAN ( $D, Eps, MinPts$ )
// All objects in  $D$  are unclassified.
begin
  FORALL objects  $o$  in  $D$  DO:
    IF  $o$  is unclassified
      call function expand_cluster to construct a cluster wrt.  $Eps$  and  $MinPts$ 
      containing  $o$ .
    end
  end

FUNCTION expand_cluster ( $o, D, Eps, MinPts$ ):
begin
  retrieve the  $Eps$ -neighborhood  $N_{Eps}(o)$  of  $o$ ;
  IF  $|N_{Eps}(o)| < MinPts$  // i.e.  $o$  is not a core object
    mark  $o$  as noise and RETURN;
  ELSE // i.e.  $o$  is a core object
    select a new cluster-id and mark all objects in  $N_{Eps}(o)$  with this current cluster-id;
    push all objects from  $N_{Eps}(o) \setminus \{o\}$  onto the stack seeds;
    WHILE NOT seeds.empty() DO
      currentObject := seeds.top();
      retrieve the  $Eps$ -neighborhood  $N_{Eps}(currentObject)$  of currentObject;
      IF  $|N_{Eps}(currentObject)| \geq MinPts$ 
        select all objects in  $N_{Eps}(currentObject)$  not yet classified or are marked as noise,
        push the unclassified objects onto seeds
        and mark all of these objects with current cluster-id;
      seeds.pop();
    RETURN
  end
end

```

Figure 2: DBSCAN Algorithm

To find a cluster, DBSCAN starts with an arbitrary object  $p$  in  $D$  and retrieves all objects of  $D$  density-reachable from  $p$  with respect to  $Eps$  and  $Minpts$ . If  $p$  is a border object, no objects are density-reachable from  $p$  and  $p$  is assigned to noise temporarily. Then DBSCAN handles the next object in database  $D$ . Retrieval of density-reachable objects is performed by successive region queries. A region query returns all objects intersecting a specified query region efficiently by R\*-trees. Before clustering the database, R\*-tree should be built in advance. However, there are some DBSCAN algorithm problems limiting its applications.

**The most fundamental open problem is:** DBSCAN requires the user to specify a global threshold  $Eps$  ( $Minpts$  is often fixed to 4 to reduce the computational amount). In order to determine  $Eps$ , DBSCAN has to calculate the distance between an object and its  $k$ th ( $k=4$ ) nearest neighbor for all objects. In addition, DBSCAN is based on R\*-tree, and calculates the  $k$ -dist value on the entire database. The two procedures are the most time-consuming phases in the whole clustering process, but their computational loads are not included in time consumption as in  $O(n \log n)$ , so the actual time consumption of DBSCAN may be larger than that of  $O(n \log n)$ [15]. Clustering procedure is very expensive so that it is computationally prohibitive for large databases.  $Eps$  and  $Minpts$  determine a density threshold, thus DBSCAN becomes a typical density-based clustering method. Furthermore, the  $Minpts$  usually is fixed to 4, thus the density threshold is perfectly determined by  $Eps$ . The DBSCAN algorithm is sketched in Figure 2.

#### 4. Data space partitioning into overlapped circles

The key idea of the proposed algorithm is that one can greatly reduce the number of distance computations required for clustering by partitioning the data into overlapping subsets, and then only measuring distances among pairs of data points that belong to a common subset. The

overlapped circle technique thus uses two different sources of information to cluster items: a cheap and approximate similarity measure (the radius of the overlapped circles that cover all data space) and a more expensive and accurate similarity measure (the optimal value of Eps).

The proposed algorithm divides the clustering process into two stages. In the first stage, the algorithm use the cheap distance measure in order to creates some number of overlapped circles (or hyper sphere), any circle is simply a subset of the elements (i.e. data points or items) that, according to the approximate similarity measure, are within some distance threshold from a central point. Significantly, a data point may appear under more than one circle, and every data point must appear in at least one circle. The circles with solid outlines in Figure 3 show an example of overlapping circles that cover a data set. Note that every solid circle (contains nearest adjacent for a central point) is a subset of larger dashed circle (contains nearest far adjacent for a central point). Dashed circles are used to ensure that no cluster split into more than one cluster. If you look at cluster 1, this cluster split into two clusters without dashed circle because there is no data point in the intersection of the two inner solid circles, using the dashed circle when it is required, the algorithm detects the actual clusters contained in data set.

In the second stage, the proposed algorithm executes DBSCAN clustering algorithm, the value of Eps is obtained from the overlapped circular regions, thus the proposed algorithm does not require any input parameter.

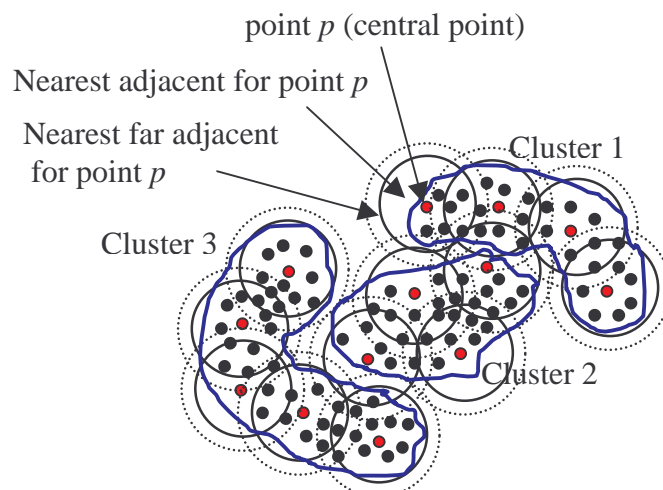


Figure 3: An example of three data clusters and the circles that cover them.

#### 4.1 Creation of overlapped circles

The proposed algorithm computes the radius ( $Rad$ ) of circles that cover all data points as we will see in the next section. Figure 4 illustrates the creation of overlapped circles. The function takes the first point as the center of the first circle (step 2 of Figure 4), and assign all points whose distances from this center are less than or equal to the value of  $Rad$  to the list1 (i.e. the list1 contains nearest adjacent for the central point of the circle as in step 6). If a point  $p$  is covered by more than one circle then  $p.distance$  keeps it's distance to the nearest circle (step 8), also  $p.circle$  keeps the identification of the nearest circle as in step 9. The function assign all points whose distances from this center is larger than  $Rad$  and less than  $Rad * 1.5$  to the list2 as in step 12 (i.e. points which lie between the solid and dashed circle are the nearest far adjacent for the central point of the circle as shown in Figure 3). For each point in list1, the algorithm keeps the distance to the center of the nearest circle and the identification number of that circle as in step 8 and step 9. The center of the next circle is the point whose distance is larger than  $Rad * 1.5$  and less than or equal to  $Rad * 2$  from the center of current circle to ensure from the existence of overlapping (this is shown in step 14). This process continues tell all points are covered. Steps from 20 to 22 search for uncovered points remaining to cover them by creating new circles.



Thus every circle contains two list; the first list contains all points inside the solid edge (nearest adjacent for a central point as in Figure 3), the other list contains all points outside the solid edge and inside the dashed edge (nearest far adjacent for a central point as in Figure 3). Only, the algorithm uses the points inside the solid edge to find maximum of (minimum pairs wise distance) to compute the optimal value of Eps, that will be used by the DBSCAN algorithm in the next stage.

When the algorithm retrieves the neighbors of a point, it directly goes to the best (nearest) circle that covers the point (a point may be covered by more than one circle), computes how far is it from all the points in that circle, and returns points at distances less than or equal to Eps. If the distance of that point to the solid edge is less than Eps then the algorithm computes the distances between that point and all the points contained in dashed circle (points in list1 and list2), retrieves the points in Eps distance, and the point is assigned to current cluster if it is a core point, or assigned noise temporally.

```

Function Creat_circles( $D$ ,  $Rad$ )
  //creat all required overlapped circles that cover all data points
  //all points in  $D$  are uncovered
begin
1. circle_id=0
2. circle_center[circle_id] = the first point in  $D$ 
3. next_center = false
4. for all point  $p$  in  $D$ 
5.   if (distance ( $p$ , circle_center[circle_id]) $\leq Rad$  )
6.     { push point  $p$  to linked list1 of the circle_id
7.       if  $p$ .distance > distance ( $p$ , circle_center[circle_id])
8.          $p$ .distance = distance ( $p$ , circle_center[circle_id])
9.          $p$ .circle= circle_id
10.    }
11.  else if (distance ( $p$ , circle_center[circle_id]) $\leq Rad*1.5$  )
12.    push point  $p$  to linked list2 of the circle_id
13.  else if (distance ( $p$ , circle_center[circle_id]) $\leq Rad*2$  )
14.    and ( $p$  is uncovered)
15.    { circle_center[circle_id + 1] =  $p$ 
16.      next_center = true
17.    }
18.  if (next_center == true)
19.    circle_id ++ , goto step 3
20. endfor
21. for all points  $q$  in  $D$ 
22.  if  $q$  is uncovered point
23.    circle_center[circle_id++] =  $q$ , goto step 3
end function

```

Figure 4: Creation of circles that cover all data points

## 5. The Proposed Algorithm

The proposed algorithm merges ideas from many algorithms. It is based on DBSCAN while we try to solve the problems of Eps and R\*-tree. in this section, the R\*-tree is discussed, how the value of Eps is computed, and the computation of the radius of all data space.

### 5.1 R\*-tree and overlapped circles

What is R\*-tree?. What is the problem of it?. The R\*-tree [1] generalizes the 1-dimensional B-tree to d-dimensional data spaces, specifically an R\*-tree manages d-dimensional hyper rectangles instead of 1-dimensional numeric keys. An R\*-tree may organize objects such as polygons using

minimum bounding rectangles (MBR) as approximations as well as point objects as a special case of rectangles. The leaves store the MBR of data objects and a pointer to the exact geometry. Internal nodes store a sequence of pairs consisting of a rectangle and a pointer to a child node. These rectangles are the MBR's of all data or directory rectangles stored in the subtree having the referenced child node as its root (Figure 5). To answer a region query, we start from the root, and the set of rectangles intersecting the query region is determined and then their referenced child nodes are searched until the data pages are reached (i.e. more than one path is searched) and the search space become large. Since the overlap of the MBR's in the directory nodes grows with increasing dimension  $d$ , the R\*-tree is efficient only for moderate values of  $d$  [4].

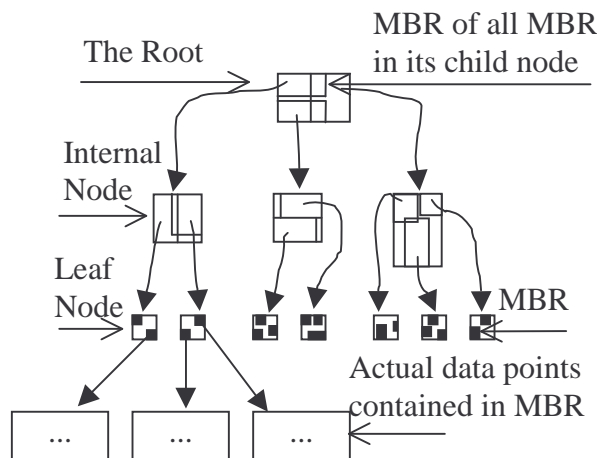


Figure 5: Sample R\*-tree

We partition the data space into overlapped circular (sphere or hyper sphere) regions such that the radius of each circle is larger than the expected  $\epsilon$ . We use these overlapped circles to answer a region query instead of R\*-tree. This idea comes from Figure 1, the overlapped circle of the same radius may be used to cover all data space with respect to the radius of circle is greater than the expected  $\epsilon$ . Some data points may be belonging to more than one circle but we use the nearest circle to retrieve it's neighborhood using Euclidean distance. This search time is better than that of R\*-tree, since in R\*-tree to answer a region query, we start from the root, and the set of rectangles intersecting the query region is determined and then their referenced child nodes are searched until the data pages are reached (i.e. more than one path is searched) and the search space become large. In the other hand, in the proposed algorithm say  $k$  of circles cover all data space. Thus, the search requires  $O(mn)$ , where  $n$  is the number of data points,  $m=n/k$  is very small compared with  $n$ . For each point we keep the nearest circle center and the distance to it. The radius of the circle depends on the radius of all data space.

### 5.2 Computation of the Radius of the data space and $\epsilon$

How does the proposed algorithm determine the radius of all data space ?. The proposed algorithm accumulates all data points in a single cluster called cluster feature (CF); CF is a data structure summarizing information about all points in the dataset,  $CF = (n, \vec{LS})$ , where  $\vec{LS}$  is the linear sum of the  $n$  data points i.e.  $\vec{LS} = \sum_{i=1}^n \vec{x}_i$ ,  $n$  is the number of points in dataset,  $\vec{x}_i$  is a  $d$ -dimensional data point. The center of the all data points in the dataset is

$$\vec{x}_0 = \vec{LS} / n = \sum_{i=1}^n \vec{x}_i / n. \text{ The radius of the data space is given by } R = \sqrt{\sum_{i=1}^n (\vec{x}_i - \vec{x}_0)^2 / n}.$$

So,  $R$  is the radius of the circle (sphere) that contains all data points in our dataset. we compute the area of that circle from the relation “area =  $3.14 * R^d$ ” (we refer to this area as circular area), then we compute the area from other view, by using minimum bounding rectangle that contains all data

points in the dataset, “area =  $\prod_{i=1}^d L_i$ ”, where  $d$  is the dimension of the points,  $L_i$  is the length of dimension  $i$ , which is equal to the difference between the maximum and the minimum value in dimension  $i$  (we refer to this area as rectangular area). Figure 6 shows the circular area and rectangular area of some data points in 2 dimension.

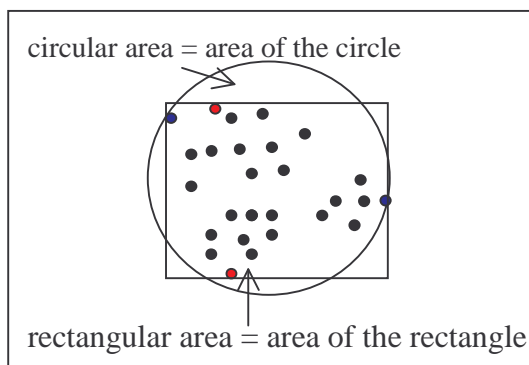


Figure 6: Circular and Rectangular area of data space

In Figure 6, the length of the first dimension is determined by the two blue points (  $x$  dimension ). The length of the second dimension is determined by the two red points (  $y$  dimension ).

The proposed algorithm partitions data space into overlapped circles. The radius of these circles should be depends on the dimension of data space, since as the dimension increase the data points will be more sparse. Also, the radius should be depends on the area of data space, but which area we can use?. Experimentally the ratio between the two area is the best, so ratio area = (circular area / rectangular area) or the inverse i.e. ratio area = (rectangular area / circular area). The ratio area should be less than or equals to one i.e.  $0 < \text{ratio area} < 1$ . Experimentally the best relation for the radius of the overlapped circles is  $Rad = d * \text{ratio area} + \text{ratio area} / 2$ , where  $d$  is the dimension of the data space, ratio area is the ratio between circular area and rectangular area or the inverse. As the dimension increase the radius of the overlapped circles increase. Also, as the difference between the two areas decrease the ratio area increase and the radius of the overlapped circles increase. Always, the radius of the overlapped circles is greater than the expected Eps.

Here, we are ready to apply DBSCAN, but we will use circles that cover all data space to calculate the optimal value for Eps. To do this, for each point in the solid circle we find the distance to its nearest neighbor, then we keep the distance between the far nearest pairs, we perform this process for all circles, this process is equivalent to distances matrix update in the single link algorithm. The following example explain the idea.

**Example 1:** suppose one of the overlapped circles contains the following six points in 2 dimensional space.

Table1: Six points in 2 dimension

	Attribute1	Attribute2
<b>P1</b>	1	1.1
<b>p2</b>	1.25	0.9
<b>P3</b>	1	1.25
<b>P4</b>	1.23	1.22
<b>P5</b>	1.12	1.5
<b>P6</b>	1.13	0.7

Then all pairs wise distances are in table 2 (distances matrix ) these distances are calculated by using the next equation.

$$d(p_i, p_j) = \sqrt{\sum_{k=1}^d (p_{i,k} - p_{j,k})^2}$$



Note that, all six points are in list1 i.e. points are inside the solid edge only.

Table 2: Pairs wise distances (distances matrix )

	P1	P2	P3	P4	P5	P6
P1	0.000	0.320	0.150	0.259	0.418	0.421
P2	0.320	0.000	0.430	0.321	0.614	0.233
P3	0.150	0.430	0.000	0.232	0.277	0.565
P4	0.259	0.321	0.232	0.000	0.301	0.530
P5	0.418	0.614	0.277	0.301	0.000	0.800
P6	0.421	0.233	0.565	0.530	0.800	0.000

From table 2 we find the nearest neighbor for each point is as follow:-

- P1 & p3 at 0.150
- P2 & p6 at 0.233
- P3 & p1 at 0.150
- P4 & p3 at 0.232
- P5 & p3 at **0.277**
- P6 & p2 at 0.233

The nearest pair are p1 and p3 at 0.150. The far nearest pair are p5 and p3 at **0.277**.

For this circle, the maximum distance between the nearest pair is 0.277 that is between p3 and p5. So the algorithm keeps this distance, then the algorithm take the next circle, this process is performed for all overlapped circles. Then Eps will be the average of these distances. i.e.  $Eps = \sum max_i / k$ , where  $k$  is the number of created overlapped circles and  $max_i$  is the distance between the far nearest pair of points in circle number  $i$ .

we use Eps as calculated before to overcome the presence of outliers. In the proposed algorithm the Minpts is fixed to 3 by the experiments.

Why Minpts is fixed to 3? 3 points is the best number for Minpts according to the method that we used to determine the value of Eps, if you examine the points in example 1, you find that all the six points are outliers

When do we use the points in list2 (points between solid and dashed circle)?. To answer this question look at the following Figure 7,

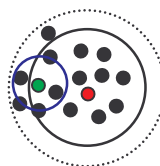


Figure 7: Neighbors of the green point wrt. Eps.

Suppose that  $Rad = 1$ ,  $Eps = 0.4$ , distance between the green point and the red point (the center) = 0.8, then the distance between the green point to the solid edge = 0.2 and this distance is less than Eps. So, in this case we calculate the distance between the green point and all points in dashed circle to find its neighbors wrt. Eps ( points inside the blue circle ).

After determination of Eps we apply the basic process of DBSCAN. We can summarize the basic process of the proposed algorithm in the following steps:

- 1- find the center of all data.
- 2- calculate the average radius of data (radius of circle that covers all data points).
- 3- divide the data into overlapped circular regions of the same radius such that this radius is larger than the expected Eps.
- 4- In each region, compute all pair wise distance (distance matrix as in Slink algorithm), find

the maximum of (minimum distances). If we have  $k$  circles cover all data there will be  $k$  values for distance, we take the average value of these  $k$  distances to present the Eps of DBSCAN algorithm.

- 5- Apply DBSCAN algorithm on data using Eps obtained from step 4, to retrieve the neighbors of point  $p$ , only distances between point  $p$  and all points share the same region are calculated. This data structure is better than R\*-tree, since in R\*-tree more than one path is traversed and R\*-tree works well with low dimensional data.

The proposed algorithm is the same as shown in Figure 2, while Eps is not user input parameter, and overlapped circular region is used to answer a region query instead of R\*-tree.

## 5.1 Complexity

As we discussed before, the proposed algorithm composed of two stage, in first stage, the algorithm creates  $k$  circular regions cover the data space, this requires  $O(nk)$ , where  $n$  is number of data point. To find the Eps the algorithm find pair wise distance in each region, in average each region contains  $m$  points, where  $m = n/k$ , thus this process takes  $O(m^2k)$ . So the time of first stage is  $O(nk + m^2k)$ . In second stage, the algorithm apply the DBSCAN, using the circular regions to answer query region. The search for the points in Eps distance from a random point requires  $O(m)$ , thus the DBSCAN requires  $O(nm)$ . Hence the total time complexity of the proposed algorithm is  $O(nk + m^2k + nm)$ , where  $k$  is the number of circles cover the data space,  $m$  is the average number of points in each circle,  $m = n/k$  and  $n$  is number of points in the data set.

## 6. Experimental Results

We evaluated the proposed algorithm on several different real and synthetic datasets. We compared our results with that of DBSCAN algorithm in terms of the total quality of clusters, both algorithms produce the same result. Our experimental results are reported on PC 800 MHZ, 128 MB RAM, 256 KB cache. we give a brief description of the datasets used in our algorithm evaluation. Table 3 shows some characteristics of the datasets.

Table 3: Characteristic of the datasets

Datasets	Number of records	Number of attributes	Type of dataset
Earthquake	2049	2	real
Abalone	4177	7	real
Wind	6574	15	real
Db1	10000	2	synthetic

In Table 4, we present the results obtained from the proposed algorithm, present the optimal value for Eps that is always less than the radius of circle (Rad), number of circles present total number of circle that cover all data points. This number is equivalent to the number of leaf nodes in R\*-tree. But in case of R\*-tree we can not directly reach the required leaf node. We may also need to reach to more than one leaf node, this problem grows as the dimension of data grow. In the proposed algorithm only one circle is directly reached, this process saves time.

Table 5 presents the results obtained from the DBSCAN algorithm. The input values for Eps parameter is the same as in the proposed algorithm. Comparing the results of the proposed algorithm and DBSCAN algorithm, both algorithm produce the same results nearly.

Table 4: Results of the proposed algorithm

Datasets	Time (sec.)	Radius of all data	Number of circles	Rad	Eps	Number of clusters	Number of noise points
Earthquake	1	0.599	27	0.317	0.124	6	35
Abalone	14	0.489	30	0.245	0.152	2	28
Wind	31	19.17	920	9.584	8.54	12	704
Db1	4	5.527	95	1.224	0.249	100	4

Table 5: Results of the DBSCAN algorithm

Datasets	Time (sec.)	Eps	clusters	noise
Earthquake	3	0.1238	6	35
Abalone	24	0.1520	2	28
Wind	116	8.5401	11	703
Db1	58	0.2485	100	4

Figure 8 shows a comparison between DBSCAN and the proposed algorithm, which demonstrates the efficiency of the proposed algorithm.

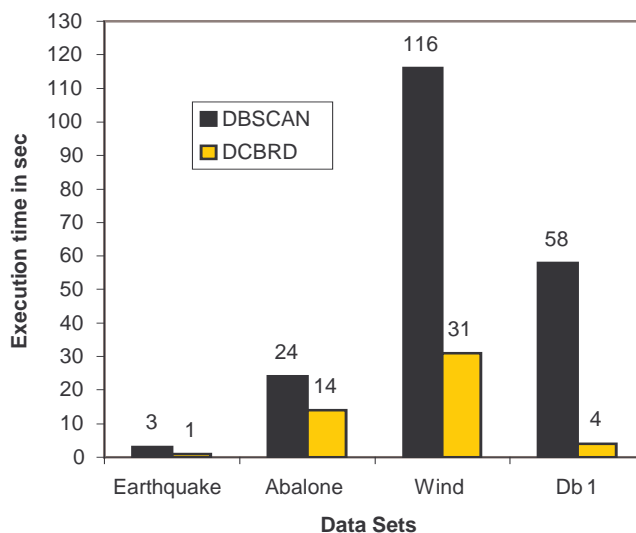


Figure 8: Execution time

## 7. Conclusion

In this paper, we presented a density based clustering algorithm require no input parameters. The proposed algorithm handles large data set efficiently and discover any arbitrary shaped clusters of any size. This algorithm is based on partitioning the data into overlapped circular or hyper spherical regions and use the best region to retrieve the neighborhood of any data point. Our experimental results demonstrated the efficiency of the proposed algorithm.

## References

- [1] Beckmann N., Kriegel H.-P., Schneider R, and Seeger B. 1990. The R\*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.
- [2] DEFAYS, D. An efficient algorithm for a complete link method. The Computer Journal, vol. 20, 1977, pp. 364-366.
- [3] Ester M., Kriegel H.-P., Sander J., Xu X.: “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, 1996, pp. 226-231.
- [4] Ester M., Kriegel H-P., Sander J., Xu X. “Clustering for Mining in Large Spatial Databases”, Special Issue on Data Mining, KI-Journal, ScienTec Publishing, Vol. 1, 1998, pp. 1-7.
- [5] Fayyad U., Piatetsky G., Smyth P., Uthurusay R., “Advances in knowledge discovery”, AAAI press, Cambridge,1996.
- [6] Gordon A. D. “A review of hierarchical classification”, Journal of the Royal statistical society. Series A, Vol.150, 1987, pp. 119-137.
- [7] Guha S., Rastogi R., Shim K.: ”CURE: An Efficient Clustering Algorithms for Large Databases”, Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle, WA, 1998, pp. 73-84.
- [8] HAN J., KAMBER M., and TUNG A. K. H. Spatial clustering methods in data mining: A survey. In Miller, H. and Han, J. (Eds.) Geographic Data Mining and Knowledge Discovery, Taylor and Francis, 2001.
- [9] Krznaric D. and Levkopoulos C.. Optimal algorithms for complete linkage clustering in d dimensions. *Theor. Comput. Sci.*, 286(1), 2002, pp. 139–149.
- [10] Kriegel. H-P., Peer K., and Irina G., “Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering.”, Proc. 5th Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK'03), Prague, Czech Rep., 2003, pp. 224-233.
- [11] Kaufman L., Rousseeuw P. J.: “*Finding Groups in Data: An Introduction to Cluster Analysis*”, John Wiley & Sons, 1990.
- [12] MacQueen, J.: “*Some Methods for Classification and Analysis of Multivariate Observations*”, 5th Berkeley Symp. Math. Statist. Prob., Vol. 1, 1967, pp. 281-297.
- [13] Ng R. T., Han J.: “*Efficient and Effective Clustering Methods for Spatial Data Mining*”, Proc. 20th Int. Conf. On Very Large Data Bases, Santiago, Chile, Morgan Kaufmann Publishers, San Francisco, CA, 1994, pp. 144-155.
- [14] Sibson R.: “*SLINK: an optimally efficient algorithm for the single-link cluster method*”. The Computer Journal, Vol. 16, No. 1, 1973, pp. 30-34.
- [15] Shi-hong Y.,Ping L., Ji-dog G.and Shui-geng Z.”A Statistical Information-based clustering Approach in distance space”, JZUS, vol. 6A(1), 2005, pp. 71-78.
- [16] Voorhees, E.M. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing and Management*, 22, 6, 1986, pp. 465-476.
- [17] YUE S., LI P., GUO J. and ZHOU S. “A statistical information-based clustering approach in distance space” *J Zhejiang Univ SCI* ,vol.6A(1) , 2005, pp.71-78.
- [18] Zhang T., Ramakrishnan R., Linvy M.: “*BIRCH: An Efficient Data Clustering Method for Very Large Databases*”. Proc. ACM SIGMOD Int. Conf. on Management of Data, ACM Press, New York, 1996, pp.103-114.

---

Article received: 2006-08-19